August 2019

EMPATHY BASED REINFORCEMENT LEARNING

Dhwani Himanshu Patel
Syracuse University

ABSTRACT

Among the different emotional functions, empathy is difficult to model but is important for a robot to have so that they can socially interact with society. Perceived empathy has positive consequences on attitude and the social behaviors of an individual. The project aims to model empathic behavior and perceptive capabilities in a robot in such a way that they can engage in empathic interactions with other agents in a shared physical space. In this study, a small grid environment world was designed and developed with multiple agents. Also, The Markov Decision Process and a Hand Coded algorithm were implemented and compared to study the influence of empathy on the robot agent's behavior. In this environment when em*pathy*, *needy* and food parameters are varied from -1 to 1, -1 to 1, and unlimited to limited supply, respectively, change is observed in the behavior of the robot agent. When the food supply is unlimited, the robot agent can either aid or avoid other agents based on the positive, zero or negative values of *empathy* and *needy*. In contrast, when food supply is limited, the robot agent's behavior towards other agents varies based upon the greater of the two values.

EMPATHY BASED REINFORCEMENT LEARNING

by

Dhwani Himanshu Patel

B.E., Gujarat Technological University, India, 2013
M.E., University of Massachusetts Lowell, 2015

Thesis

Submitted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

Syracuse University

August 2019

ACKNOWLEDGEMENTS

First, I would like to thank my parents Anjana Patel and Himanshu Patel. Through my journey they were a great support and motive for me to succeed. Many thanks to my elder sister Aarohi Patel and brother in law Chintan Panirwala for their support.

I would like to have special thanks to my Advisor Dr. Garrett Katz for his ongoing help, attention, and guidance kept me going forward even during the rough times. During my one year of working with him at graduate school I have learned a lot. My graduate school journey was not an easy one, but I am proud of what I have achieved thus far.

I would like to acknowledge my committee members: Dr. Chilukuri Mohan, Dr. Qinru Qiu and Dr. Sucheta Soundarajan. Thank you all for your feedback during my defense, it has helped me to greatly improve the quality of my final thesis.

Special thanks to my friends Honey Rao and Marissa Walters.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1MOTIVATION

Artificial intelligence systems have become essential aspects of our lives. AI systems have aided with daily activities and even improving upon the physical and emotional health assistance. Although they cannot feel and express empathy, it is possible to build Robots that appear to show empathy. AI Systems with empathy can empathize with a user and vice-versa. Empathetic interactions are important for the coexistence of humans and Robots in today's society, even more so for social robots, which are expected to soon emerge throughout society.

Empathy is a parameter that is difficult to define in the world of Artificial Intelligence, but it is even more difficult and complicated to quantify. Various definitions of empathy are discussed in the next section (Related Work). In this study, complementary issue of AI systems with empathy is addressed where in a tool that can quantitatively compare pre-specified imperfect formalizations of empathy in terms of agent's behavior is built. The following are the applications for which AI systems with empathy can be used:

Education: the study shows that if a tutor and student have a positive relationship, learning becomes more efficient for the student. In the field of education, learning improves if the tutor robot is intelligent and has a positive social connection

with the student and even more so if they respond empathetically to students [4].

Social Chatbots: Every website utilizes chatbots with a basic goal of collecting data and assisting customers. However, customers utilize chatbots with an expectation of a solution that has an emotional consideration. Empathetic chatbots improves customer's experience significantly, leading to customer satisfaction [5].

Assisted Living: Using social robots is a latest trend in the Ambient Assisted Living, because of their ability to involve users in human-like discussions. Social robots appear to show empathy and feelings while communicating with the individuals in Ambient Assisted Living environments [7].

Autonomous Driving: Autonomous vehicle that are equipped with emotions require interactions not only between the cars and their drivers/passengers, but also with onboard machines, neighboring pedestrians, other car and their occupants, as well as the surrounding infrastructure [6].

Patient Care: Once emotional bonds are formed between AI robots and dementia patients, there is a positive impact regarding relief of pain and suffering. For example, Paro, a robotic baby harp seal was developed in Japan by Takanori Shibati which is equipped with 32-bit processors, microphones, and several tactile sensors [8]. The robot has been found to have positive psychological effects on patients, thereby reducing stress [9].

Empathy contributes to collaboration and cooperation, and transcends cultures, social behaviors and wellbeing [14]. When building empathetic AI systems, the important question is determining the optimal action taken by the system for a specific customer, given a set of metrics or factors. The goal is for the customer to obtain some value from an interaction with

the AI system. An AI system with the capacity for empathy could provide more natural interactions, with the consideration of the customer's emotions and providing better customer satisfaction.

In this study, experiments are conducted that suggest a change will be observed in the social behavior or sensitivity of an agent/robot towards other agents based on a pre-determined empathetic level of that agent.

1.2 PROBLEM STATEMENT

As a first step towards robots with more sophisticated forms of empathy, in this thesis we are aiming to understand the impact of empathy on the behavior of the robots towards other agents/humans. In this study we use a computer simulation of a small multi-agent system consisting of 3 agents where one of the agent have varying degrees of *empathy* and *needy* toward the others. This would be the first step towards a more realistic model of empathetic agents, which is important for us to build robots with empathy. For the current system, the use of the word altruism instead of empathy might be more appropriate, but in this study, word empathy is used for the current system since the word empathy would be more appropriate once we have extended our system by implementing algorithms such as inverse reinforcement and plan recognition. In this study small grid world environment is created with 3 agents and food, in order to study how the pre-determined empathetic level of robot agent determines if it will help other agents to complete their task or whether it will only focus to complete its own task. In this environment, we created a task where agents have to eat food. Eating food is a concrete example of a more general idea of resources that a robot or human needs. In our current simulation of small multi-agent system, two agents beside the

robot agent in the environment have a fixed policy where they do not wander around in order to find food by themselves unless food or the robot agent is in the neighboring cell. This is because if we allow two agents other than the robot agent to have random movements, then food will get easily revealed to them and we won't get to study the impact of empathy in the robot agent's behavior towards other two agents, we are planning to add random movements in future when we extend our current system. In our current simulation, we are varying the numerical value of *empathy* and *needy* from -1 to 1. Zero *empathy* or *needy* value means that the robot agent does not have an empathetic or needy feeling. When *empathy* and *needy* value is varied between 0 and 1, we can study robot agent's behavior towards other agents when they have an empathetic or needy feeling. In this study, the robot agent's behavior for negative *empathy* value, is observed in order to model agents where they actively try to work against other agents. Robot agent's behavior for negative *needy* value is observed in order to model agents where they behave against their own interests. When the current system is extended and implemented with Markov Games studying negative *needy* values for the robot agent will be steppingstone to study negative *needy* values for the agents other than robot agent. In this study, we used the Markov Decision Process (discussed in detail in the Background section) to understand impact of empathy in reinforcement learning. The Hand coded policy uses Dijkstra's algorithm to understand behavior of agent with empathy. Both Markov Decision Process and Hand coded policy provide two similar yet distinct ways of formalizing empathy in an agent.

Chapter 2

RELATED WORK

2.1 EMPATHY

Artificial empathy is defined as the development of AI systems such as companion Robots, that can detect and respond to human emotions. The definition of empathy from Merriam Webster [18] is

> "the action of understanding, being aware of, being sensitive to, and vicariously experiencing the feelings, thoughts, and experience of another of either the past or present without having the feelings, thoughts, and experience fully communicated in an objectively explicit manner."

Justin Bariso in his article [15] mention that according to psychologists Daniel Goleman and Paul Ekman there are three types of empathy: cognitive empathy, emotional empathy and compassionate empathy.

> "Cognitive empathy is the ability to understand how a person feels and what they might be thinking. Cognitive empathy makes us better communicators, because it helps us relay information in a way that best reaches the other person. Emotional empathy is the ability to share the feelings of another person. Some have described it as 'your pain in my heart.' This type of empathy helps you build emotional connections with others. Compassionate empathy goes beyond simply understanding others

> and sharing their feelings: it actually moves us to take action, to help however we can.” [15].

Artificial Empathy is compassionate empathy where robots understand another individual’s feelings and respond accordingly.

2.2 RESEARCH IN AI WITH EMPATHY

In academia, robots have been created that are not only intelligent but also have social connection and demonstrate empathy towards students [4]. In this paper, author has used the EMOTE approach for Embodied Empathic Virtual and The Robotic Tutors which addresses the unique challenges of learning interactions and the social emotional cues between students and the tutor. The embodied tutor will feedback to the learning environment and further influence the tutor’s guidance. Based on tabletop learning situations where students are interacting with the Emy’s The Robotic tutor, students have a positive impact when the robot shows empathy towards them.

XiaoIce is social chatbot from Microsoft which is designed such that it establishes emotional connection with the customer(human) providing good customer service [5]. In this paper, the author uses a model that has an intelligent quotient and emotional quotient which are taken into consideration for the Markov Decision Process, used for making decisions that give optimal long term results. Results show that XiaoIce social chatbot achieves an average of 23 Conversation turns per customer session, which is higher than any other chatbots.

For autonomous vehicle [6] this paper creates framework which consists of the Perception Unit, the Emotions Unit, the Decision-making Unit, and the Decision Implementation Unit. The Emotion Unit basically tells how the model/vehicle should respond emotionally to its environment. The Emotion Unit adopts a theory proposed by risk-avoidance models. This framework is under development to be used for two-lane highway driving.

For Ambient Assisted living environment with Artificial Empathetic Robot [7], this paper uses the approach where their NICA architecture consists of a conversational agent with mind that decides which goals to implement. The agent evaluates each stage in its lifecycle, to determine whether the environment has changed. If the environment changes, then the actions and behavior of the agent also changes. The goals of the agent correspond to those of the human caregivers indicated as the most important in their daily assistance. The authors conducted experiments in which the quantitative results of the evaluation suggested that an agent's choices parallel human action for 79% of the cases in the dataset.

The findings of the study described in this paper [9] confirm the positive impact of Paro detailed in earlier work [8] and suggest that there is an even greater scope for its use in the support of residents in senior care facilities.

2.3 DIFFERENT SUB-FIELDS OF AI

There are various fields of AI that could be used for creating Robots with artificial empathy. In the Markov Decision Process, a single agent is expected to decide the best action, which is selected based on its current state [2]. Markov games are a multiagent environment model which includes multiple adaptive agents with an interacting or competing goal [12]. Inverse

reinforcement learning is the field of observing an agent's behavior, in order to learn an agent's objectives, values, or rewards [16]. Plan Recognition is the field in which an agent's top-level plans are predicted based upon its observed actions. Also, it is an abductive reasoning task that involves inferring plans that best explain observed actions [21]. The ant colony algorithm is an algorithm aimed in finding optimal paths dependent upon the behavior of ants in search of food [11]. In Co-operative game theory different agents create an alliance and help each other to maximize the result [10].

Chapter 3

BACKGROUND

3.1 MARKOV DECISION PROCESS

In AI, reinforcement learning algorithm is an algorithm where an agent learns the best action to take based on positive reinforcement. Markov Decision Process is a reinforcement learning algorithm in which an agent predicts which action is best from its current state, based on a reward. MDP (Markov Decision Process) is a model environment where an agent interacts with the environment [2].

$$\text{MDP}=\langle S,A,P,R,\gamma \rangle$$

A MDP model consists of finite number of states (S), actions (A), reward function (R), probability matrix (P) and discount factor (γ). Probability matrix suggests the possibility of an agent taking an action to reach the next state from a current state. According to the discount factor, long-term rewards put greater weight on short-term rewards. A discount factor that is closer to 1 weighs distant rewards more similarly to immediate rewards. The optimal policy of an agent is determined using an optimal utility function which is obtained using a value iteration algorithm [22]. The optimal policy can also be determined using a policy iteration algorithm [22]. In MDP, the policy determines which action is best to take from current state [2]. In MDP, you do not have to consider the previous state, all decisions are made from current state.

The goal of MDP is to find best possible action to take to reach end goal and in order to achieve that we find an optimal policy. We can find an optimal Utility function by using value iteration Optimal Utility V*(s) at state, ***s***, is a sum of the total rewards that you can receive in that state and of the action that maximizes expected utility to reach next state, ***s'***. Now, we must determine which action maximizes expected utility of next state, ***s'***, which is what optimal policy $\prod$* does. The algorithm converges when there is no change in utility between two consecutive iterations. Error is the upper bound of difference between utilities of two consecutive iterations by $||V_{i+1}-V_i|| < \epsilon \frac{1-\gamma}{\gamma}$ where, γ, is the discount factor and, *i*, is the iteration [2][22].

$$V^*(s)=R(s) + max_a[g \sum_{s'} P(s' \mid s, a) V^*(s')]$$

$$\prod^*(s)=argmax_a[\sum_{s'} P(s' \mid s, a) V^*(s')]$$

Most often, MDP finds best possible actions which may not always be the most ideal.

For example: Consider the grid world picture below:

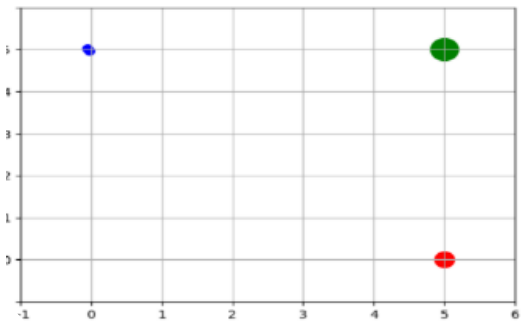

Figure 3.1 6*6 Grid World

Consider 6*6 grid world. Here, the agent's initial position is the (5,0) cell. The goal of the agent is to reach food in cell (5,5) and will be rewarded when it eats the food. Here, the agent can take 9 possible actions from the current state to go to the next state: up, down, left, right,

left-down, left-up, right-down, right-up or remain in the same state. The agent can move one step horizontally, vertically, or diagonally.

The agent receives a reward when it reaches the goal, so the optimal policy is to find the actions, with takes the least number of steps, in order to reach (4,3) cell. Two such set of actions are as follows:

- up, up, up, up, up
- left-up, up, up, up, right-up

3.2 DIJKSTRA'S ALGORITHM

Dijkstra's algorithm is a step-by-step process we can use to find the shortest path between two vertices in a weighted graph. In Dijkstra's algorithm we determine distance of all vertices in the graph from start vertex.

Dijkstra's algorithm follows the following steps [1]:

Step 1: First, initialize the distance to reach the start vertex, from the start vertex to 0. Then, set the distance from the start vertex to all other vertices in the graph as infinity. Next, create a list with all the vertices in the graph that are arranged in ascending order based on the distance from the start vertex. Mark all the vertices as unvisited.

Step 2: From the list of vertices, take vertex with the smallest distance and set that as the current vertex. We determine the distance to all vertices that are connected to current vertex, with an edge. Once the distance to all vertices that are connected to current vertex, is determined, we mark the current vertex as unvisited by removing it

from list of vertices. Store the path to reach current vertex with smallest distance in a list.

Step 3: The distance to reach the vertex from current vertex is updated if we find a path that leads to a vertex with smaller distance. Update the path.

Step 4: Repeat step 2 and step 3 until we find the smallest distance between the start vertex and all vertices in the graph. The algorithm converges when all the vertices are visited.

For example: Consider the following graph with 5 vertices connected with edges. Here, we want to find smallest distance between vertex A and vertex E. First, we initialize distance from

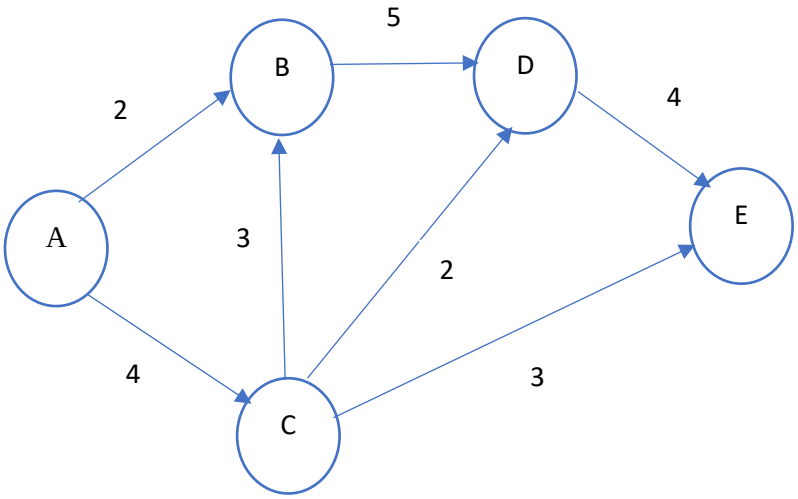

Figure 3.2 Graph

vertex A to vertex A to 0. Set the distance to all the vertices, B, C, D, and E, from A as infinity. Now vertex A has smallest distance as compared to other vertices. Set vertex A as current vertex. Vertex B and Vertex C are connected to vertex A with edge of weight 2 and 4, respectively. Now the distance from A to C is updated to 4 and distance from A to B is

updated to 2. Mark vertex A as visited. Next, the vertex with smallest distance from A, is B. Set B as the current vertex. Vertex B is connected to vertex D. The distance from A to D is updated to 7. Mark vertex B as visited. Next, the vertex with smallest distance from A, is C. Vertex C is connected to vertex B, D and E. The distance to vertex B from A via C is 7, which is greater than distance from A directly to B, which is 2. So, the distance to B is not updated. The distance to vertex D from A via C is 6, which is smaller than the distance to D from vertex A, so the distance is updated to 6. The distance to vertex E from source vertex A via C, is 7. Mark vertex C as visited. Next, the vertex with smallest distance from A is D. D is connected to vertex E. The distance from source vertex A to E via D, is 10, which is greater than Distance from A to E via C, which is 7, so the distance is not updated. Mark vertex D as visited. Vertex E is destination vertex.

Therefore, by using Dijkstra's algorithm [1] we found the shortest path from source vertex A to destination vertex E is A-C-E with distance of 7 units.

Chapter 4

IMPLEMENTATION

4.1 PROBLEM STATEMENT 1 – CAT-MDP:

The goal of this project is to observe what occurs when empathy is a factor in reinforcement learning. To gain further understanding, we formulated a small problem with 3 agents: Robot cat, subject 1 cat and subject 2 cat. To understand impact of empathy in the behavior of Robot cat towards subject 1 cat and subject 2 cat we will vary degree of *empathy* and *needy*.

Our problem involves a small grid world environment with 3 agents and food. The 3 agents in the environment are the Robot cat and a two subject cats: subject 1 and subject 2. The Robot agent gets positive reinforcement for eating. Food appears on the grid at one of four corners. The Robot agent can see the entire grid, whereas the subject 1 and subject 2 can only see nearby. The subject 1 and subject 2 have a fixed policy. If subject 1 and subject 2 see food, they go to the food, else if they see the Robot agent, they go to the Robot agent, else they stay put. The Robot agent can have varying degrees of *needy* and *empathy*. The *needy* value determines how much the Robot agent wants to eat. The *empathy* value determines how much the Robot agent wants the subject 1 and subject 2 to eat. Both can be positive or negative numbers or 0. The Robot's agent policy is to be determined by MDP value iteration algorithm [2][22]. We are going to apply Markov Decision Process [2] to this problem and see how the Robot agent, subject 1 and subject 2 behave based on *empathy* and *needy* factor.

For the purposes of this study we used a 6*6 grid world. There are two subject agents: subject 1 and subject 2, and one Robot agent. Food is placed in one of the four corners. With this experimental design, the total number of states in 6*6 grid world with subject 1 agent, subject 2 agent, the Robot agent and food appears in one, some or all the 4 corners is 746,496. All the states are assigned index in the form of i, j, k.

Let n = number of subjects

$$\text{num\_states} = [(grid\_cols * grid\_rows)^{(1+n)}] * (2)^4$$

$$= [(5 * 5)^3] * 2^4$$

$$= 250{,}000$$

$$\text{num\_states} = [(grid\_cols * grid\_rows)^{(1+n)}] * (2)^4$$

$$= [(6 * 6)^3] * 2^4$$

$$= 746496$$

4.2 MOVE FUNCTION FOR THE ROBOT, SUBJECT 1 AND SUBJECT 2

Our Implementation uses a move function to update the state when the Robot agent chooses to move dx, dy grid units. Subject 1 and subject 2 will also move according to their fixed policy. The Robot agent moves one unit at a time horizontally/vertically/diagonally. dx and dy will be in [-1,0,1]. The move function will return the new state after the agents have moved.

The Robot agent can see the entire grid, the subject 1 and subject 2 only see nearby. The subject 1 and subject 2 have a fixed policy. If subject 1 and subject 2 see food, they go to the food, else if they see the Robot agent, they go to the Robot agent, else they stay put.

Pseudocode for move function is as follow:

Algorithm 1 Move(state,dx,dy)

```
for subject 1 and subject 2 do
    foodstep = False
    for all location where food appears do
        if distance from subject to food ≤ 1 then
            foodstep: True
            new location of subject becomes current location of food (fx,fy)
            break
        end if
    end for
    if foodstep : True then
        continue
    end if
    if distance from subject to robot ≤ 1 then
        new location of subject becomes current location of robot (mx,my)
        continue
    end if
    subject stays at the same location
end for
new robot location = current robot location + (dx,dy)
new state =(new robot location,new subject 1 location, new subject 2 location, food)
return new state
```

Figure 4.1 Pseudocode for Move Function

The subject 1 and subject 2 agents can only view 1 unit far horizontally, vertically, and diagonally. So, if the distance between food and subject 1 and subject 2 is less than equal to 1 unit than subject 1 and subject 2 can see food and will move to the cell where food appears in the

next time step and will eat it. If the distance between two subjects cat agent and the Robot agent is less than equal to 1 unit than subject 1 and subject 2 can see the Robot agent and will follow the Robot agent which means that they will move to the cell that the Robot agent is currently at in the next time step. If they can neither see the Robot agent or food which means that when distance between the Robot agent and subject 1, Robot agent and subject 2, food and subject 1, food and subject 2 is greater than 1 unit than subject 1 and subject 2 will stay in same cell they are currently at in the next time step.

The Robot agent will move one unit at each time step towards food or subject 1 or subject 2 based on its optimal policy.

4.3 PROBABILITY MATRIX FUNCTION

Our Implementation also uses a function that builds the transition arrays P defining the MDP. P[(dx, dy)] is a 2d probability matrix when action (dx, dy) is taken. So P[(dx, dy)] is a 749K x 749K matrix, for our current scenario. However, most probabilities are zero because for example it is impossible for the Robot agent to jump 3 spaces. So P[(dx ,dy)] can be feasibly stored as a sparse matrix. For this we use scipy's "CSR" format to build the sparse matrices. We used function from [17] link to build sparse matrix.

P[(dx, dy)][i, j] :If the Robot agent is in state with index i, and moves by (dx, dy),this matrix element is the probability the Robot agent will end up in another state with index j.

4.4 REWARD FUNCTION

This function computes the Robot agent's reward in the given state. By default, the reward is zero. If the Robot agent location is co-located with food location, numerical value of *needy* is added to Robot agent's reward. If a subject 1 and subject 2 location is co-located with food location, the numerical value of *empathy* is added to Robot agent's reward per each subject 1 and subject 2 agents. Either *empathy* or *needy* factor can be negative. Table 1 shows the behavior of the Robot agent, subject 1 agent and subject 2 agent based on numerical value of *empathy* and *needy*.

$$\sigma_t = \begin{cases} needy\ if\ robot\ at\ food\ at\ time\ step\ t \\ 0\ otherwise \end{cases}$$

$$e_{tb} = \begin{cases} empathy\ if\ robot\ at\ food\ at\ time\ step\ t \\ 0\ otherwise \end{cases}$$

$$r_t = \sigma_t + \sum_b e_{tb}$$

Here σ_t reward is the numerical value of *needy* that the Robot agent gets when the Robot agent is at the food location at time step t otherwise σ_t reward is 0 at time step t. e_{tb} reward is the numerical value of *empathy* that the Robot agent gets when subject b is at the food at time step t otherwise e_{tb} reward is 0 at time step t. Since there are two subjects: subject 1 and subject 2, total reward that Robot agent gets is $\sum_b e_{tb}$ which is summation of rewards that Robot agent gets if subject 1 is at the food location or not and if subject 2 is at the food location or not at time step t. Total reward r_t at time step t is sum of $\sum_b e_{tb}$ (total reward that Robot agent gets if subject 1 and subject 2 is at the food location or not at time step t) and σ_t (reward that Robot agent gets if the Robot agent is at food location or not)

EMPATHY	NEEDY	THE ROBOT AND SUBJECT 1 AND SUBJECT 2 ACTION
>0	<0	The Robot agent helps subject 1 and subject 2 to find food. subject 1 and subject 2 eat food. The Robot agent will not eat food.
>0	0	The Robot agent helps subject 1 and subject 2 to find food. subject 1 and subject 2 both eat food.
>0	>0	The Robot agent, subject 1 and subject 2 both eat food.
<0	<0	The Robot agent doesn't eat the food. subject 1 and subject 2 eats food if they can see it.
<0	0	The Robot agent will try so that subject 1 and subject 2 don't eat the food. subject 1 and subject 2 eats food if they can see it.
<0	>0	The Robot agent eats food and avoids the subject 1 and subject 2. subject 1 and subject 2 eats food if they can see it.
0	<0	The Robot agent doesn't eat the food. subject 1 and subject 2 eats food if they can see it.
0	0	subject 1 and subject 2 eats food if they can see it.
0	>0	The Robot agent eats the food. subject 1 and subject 2 eats food if they can see it.

Table 4.1 Reward Table for The Robot

Above table gives an idea how for different numerical value of *empathy* and *needy* the Robot agent, subject 1 and subject 2 behaves conceptually.

4.5 VALUE ITERATION ALGORITHM

MDP value iteration algorithm is a famous classical algorithm [2] [22] that helps to learn the optimal policy for the Robot agent. Conceptually, this is an efficient way to run all possible simulations in parallel. The value iteration procedure returns a Utility vector. Utility is the net long-term reward the Robot agent can get from a given state.Net long-term rewards put more weight on short-term rewards according to the discount factor. A discount factor closer to 1 weighs distant rewards more similarly to immediate rewards. As the number of iteration increases, the closer this algorithm gets to approximating true utility. Once the utility vector is determined, the Robot agent's final policy can be determined based on the transition matrices and utility vector. In any state the Robot agent will choose the action with highest expected utility [2] [22].

Pseudocode for our implementation of the value iteration algorithm is as follow:

Algorithm 2 Value Iteration

Utility $U^0 \leftarrow 0$

error $\leftarrow \infty$

1. **for** n <iteration do :
2. **for** all state s **do :**
   $U^{n+1}(s_i) = r_i + \gamma \; max_k(\sum_{j=0}^{N} P[ijk]U^n(s_j))$
3. **end for**
   error = $(U^{n+1} - U^n)*(1- \gamma)/ \gamma$
4. **if** error <= target error **then**
   break
5. **end if**
6. **end for**
6. **return** U

$\pi = \arg max_k(P[:: k]U)$

Figure 4.2 Pseudocode for Value Iteration Algorithm

The utility vector is initially arbitrarily set to 0. Now we find Probability of going from one state to other for all action by multiplying it with utility of next state. Now we must find the action that maximizes the expected utility of the subsequent state, which is what an optimal policy should do. Now we multiply this with discount factor and add reward to that result. This will give us expected utility. Now value iteration converges when error equals to 0. The error is the upper bound of difference between utilities of two consecutive iterations by $||u_{i+1}-u_i|| < \epsilon \frac{1-\gamma}{\gamma}$ where γ is discount factor and i is iterations. The algorithm converges when no state's utility changes by much [19]. We can also stop the algorithm by making sure that algorithm runs for fixed number of iterations.

4.6 POLICY ITERATION ALGORITHM

Policy iteration algorithm is another famous classical algorithm used to find an optimal MDP policy [2] [22]. In this algorithm we first initialize policy, to a policy where the Robot will take certain action to go from one state to another. Now, policy iteration algorithm iteratively improves policy. Policy Iteration algorithm converges when there is no change in policy. We can find utility vector for each state for current policy by solving following equation:

$$U^n(s_i) = r_i + \gamma \left(\sum_{j=0}^{N} P[ij\pi(s_{ij})] U^n(s_j)\right)$$

Here, $\pi(s_{ij})$ is the action taken under current policy to reach state s_j from state s_i. Now, we can find if by taking different action can we improve the utility of the subsequent state, if the utility of the subsequent can be improved by taking different action then we change the policy. Policy

Iteration will converge when current optimal policy is equal to expected optimal policy i.e. it stops when there is no change in policy [2] [22].

Pseudocode for our implementation of the policy iteration algorithm is as follow:

Algorithm 3 Policy Iteration

Choose an arbitrary intial policy π'

n = 0
N = number of states

1. Repeat

$\pi = \pi$'

Compute Utility using π by solving linear equations

2. for i <=N **do :**

$U^n(s_i) = r_i + \gamma \left(\sum_{j=0}^{N} P[ij\pi(s_{ij})]U^n(s_j)\right)$

$\pi'(s_i) = \arg max_k(P[ijk]U^n(s_j))$

3. end for

n = n+1

4. Until $\pi = \pi$'

5. return π

Figure 4.3 Pseudocode for Policy Iteration Algorithm

4.7 MODIFIED MOVE FUNCTION (LIMITED FOOD)

Our Implementation uses a move function to update the state when the Robot agent chooses to move dx, dy grid units. Subject 1 and subject 2 will also move according to their fixed policy. The Robot agent moves one unit at a time horizontally/vertically/diagonally. dx and dy will be in [-1,0,1]. The move function will return the new state after the agents have moved.

Now when subject 1 or subject 2 moves to the cell where food is available, food will disappear from that cell in next time step. Similarly, if the Robot agent moves to the cell where food is available, food will disappear from that cell in next time step.

Pseudocode for modified move function is as follow:

Algorithm 4 Move Function(state,dx,dy)

```
for subject 1 and subject 2 do
    foodstep = False
    for all location where food appears do
        if distance from subject to food ≤ 1 then
            foodstep = True
            if current location of subject is same as food location (fx,fy then
                delete food from location (fx,fy)
            end if
            new location of subject becomes current location of food (fx,fy)
            break
        end if
    end for
    if foodstep : True then
        continue
    end if
    if distance from subject to robot ≤ 1 then
        new location of subject becomes current location of robot (mx,my)
        continue
    end if
    subject stays at the same location
end for
if current location of robot (mx,my) is same as food location (fx,fy) then
    food disppears from location (fx,fy)
end if
new robot location = current robot location + (dx,dy)
new state =(new robot location,new subject 1 location, new subject 2 location, food)
return new state
```

Figure 4.4 Pseudocode for Modified Move Function

4.8 MODIFIED REWARD FUNCTION (LIMITED FOOD)

In this modified reward function the Robot agent gets reward when any one of the 3 agents eat food. In this implementation as soon as any of the agent eats food, food will disappear from that location. So, for this implementation as soon as the Robot agent gets reward either for eating food or helping either subject 1 or subject 2 to eat food.

$$\sigma_t = \begin{cases} needy\ if\ robot\ at\ food\ at\ time\ step\ t \\ 0\ otherwise \end{cases}$$

$$e_{t1} = \begin{cases} empathy\ if\ subject\ 1\ is\ at\ food\ and\ robot\ is\ not\ at\ food\ at\ time\ step\ t \\ 0\ otherwise \end{cases}$$

$$e_{t2} = \begin{cases} empathy\ if\ subject\ 2\ at\ food\ while\ subject\ 1\ and\ robot\ are\ not\ at\ food \\ 0\ otherwise \end{cases}$$

$$r_t = \sigma_t + e_{t1} + e_{t2}$$

Here σ_t reward is the numerical value of *needy* that the Robot agent gets when it is at the food location at time step t while subject 1 and subject 2 are not at food location otherwise σ_t reward is 0 at time step t. e_{t1} reward is the numerical value of *empathy* that the Robot agent gets when subject 1 is at the food location at time step t while the Robot agent and subject 2 are not at food location otherwise e_{t1} reward is 0 at time step t. e_{t2} reward is the numerical value of *empathy* that the Robot agent gets when subject 2 is at the food location at time step t while the Robot agent and subject 1 are not at the food location otherwise e_{t2} reward is 0 at time step t. Since there are two subjects: subject 1 and subject 2 total reward that Robot agent gets is summation of rewards e_{t1} and e_{t2}. Total reward r_t at time step t is sum of e_{t1}, e_{t2} and σ_t.

Pseudocode for Modified reward function when food is limited is as follow:

Algorithm 5 Reward (state, needy, empathy)

```
for all location where food appears do
    if robot location is same as food location then
        reward = reward + needy
        break
    end if
    for subject 1 and subject 2 do
        if subject location is same as food location then
            reward = reward + empathy
            break
        end if
    end for
end for
return reward
```

Figure 4.5 Pseudocode for Modified Reward Function

4.9 EXAMPLES OF CAT-MDP (UNLIMITED FOOD)

- Consider *empathy* = 1 and *needy* = -1. Based on the reward, probability matrix, and utility, the optimal policy of the Robot is determined. Since reward is *empathy* the Robot agent decides to help subject 1 and subject 2 to eat food. (food is unlimited)

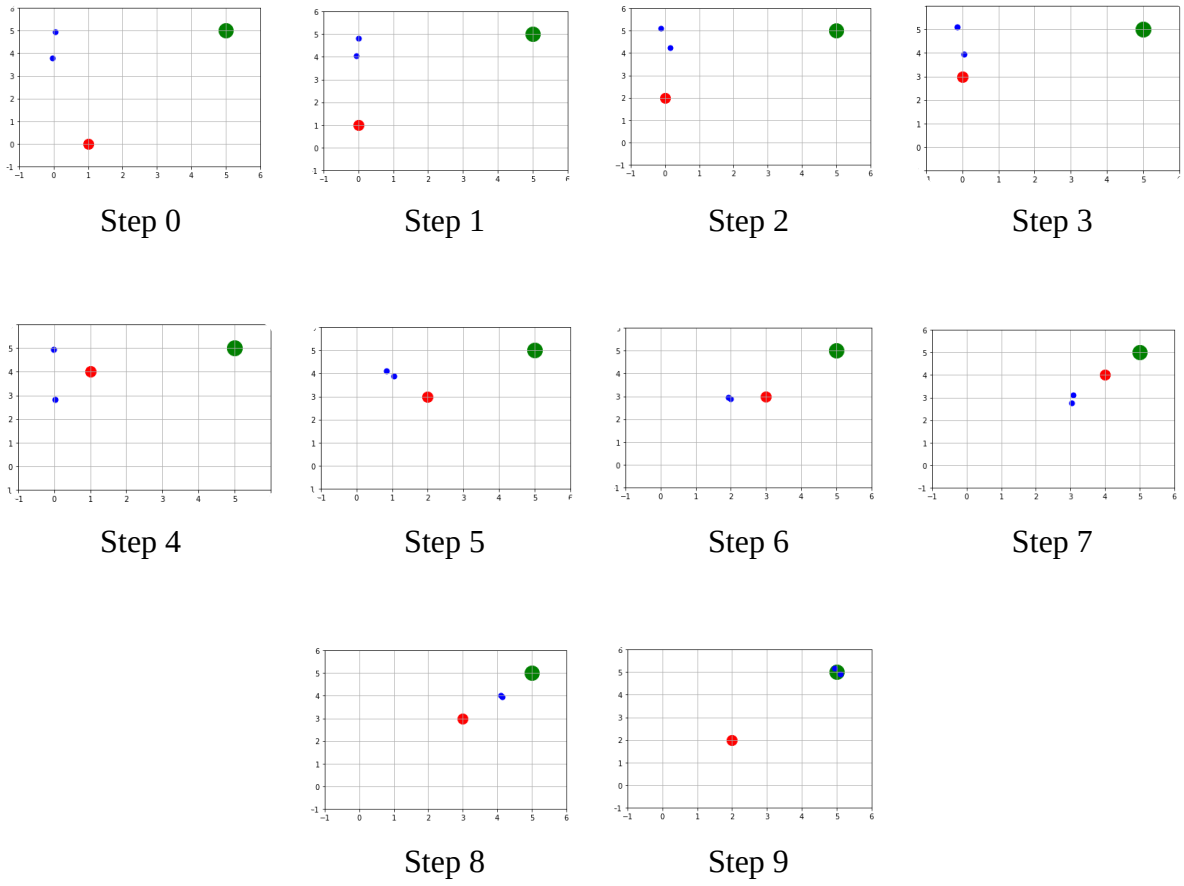

Figure 4.6 Example of CAT-MDP (Unlimited Food)

In Step 1,2,3 we can see that the Robot agent is moving towards subject 1 and subject 2 so that Robot agent can take subject 1 and subject 2 towards food. Here in step 4 one of the subject's move to the cell that the Robot agent occupied in step 3 since the Robot

agent was in one of the subject's viewing range. As you can see in step 4, the Robot agent is in viewing range of both the subject 1 and subject 2. In Step 5, both subject 1 and subject 2 would move to the cell that was occupied by the Robot agent in step 4. Similarly, in step 6,7,8 subject 1 and subject 2 would occupy the cell which was previously occupied by the Robot agent in last step. In Step 8, food is in viewing range of both subject 1 and subject 2. So, in step 9 subject 1 and subject 2 would move to the cell that has food and will eat food. Since in this scenario rewards consist of *empathy* = 1 and *needy* = -1, the Robot agent will not eat food.

- Consider *empathy* = -1 and *needy* = 1. Based on the reward, probability matrix, and utility, the optimal policy of the Robot agent is determined. Since reward is *needy* the Robot agent eats food and do not help subject 1 and subject 2. In Following steps, you can see that the Robot agent move towards the food and in step 5 and it will eat food. (Food is unlimited).

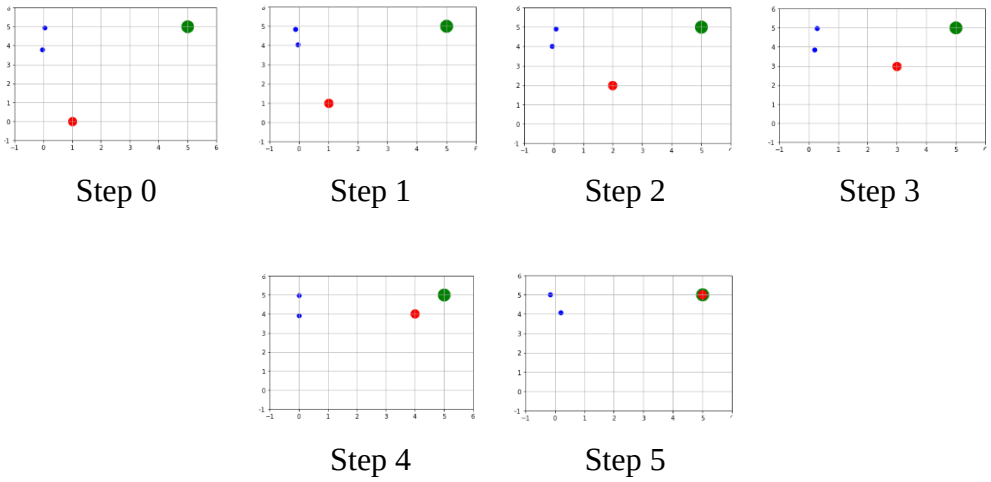

Figure 4.7 Example of CAT-MDP (Unlimited Food)

4.10 EXAMPLES OF CAT-MDP (LIMITED FOOD)

- Consider *empathy* = 0.7 and *needy* = 0.6 Based on the reward, probability matrix, and utility, the optimal policy of the Robot agent is determined. Here, food is limited. As soon as one of the agents eats the food, the food will disappear. Here, in this example, *empathy* is 0.1 greater than the numerical value of *needy* as a result the Robot agent helps subject 1 or subject 2. Here reward is *empathy* because numerical value of *empathy* is 0.1 greater than *needy*. The Robot agent help either subject 1 or subject 2 so that one of them can eat food.

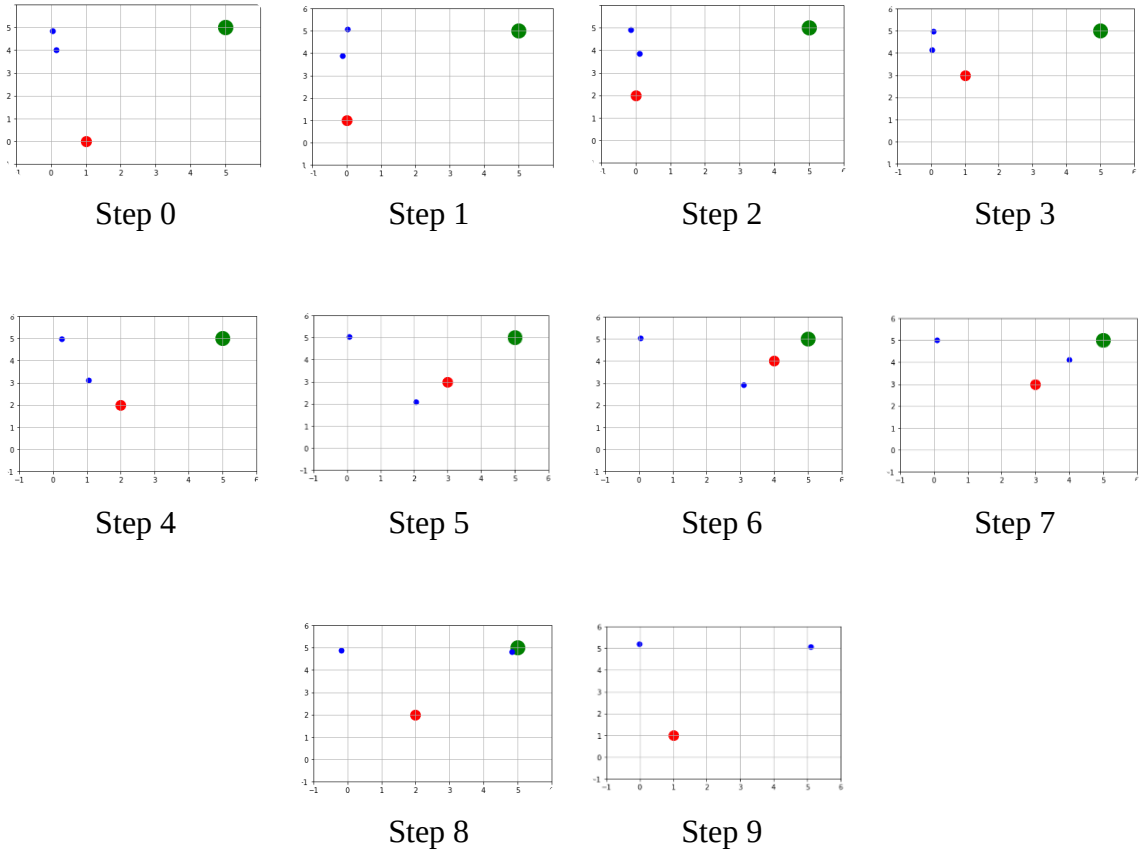

Figure 4.8 Example of CAT-MDP (Limited Food)

In Step 1,2,3 we can see that the Robot agent is moving towards one of the subject's so it can help that subject to move towards food. Here, in step 4 one of the subjects will move to the cell that the Robot agent occupied in step 4 since the Robot agent was in one of the subject's viewing range. Similarly, in step 6,7 the subject would move to the cell which was previously occupied by the Robot agent in last step. In Step 8, food is in viewing range of the subject. So, in step 9, the subject would move to the cell that has food and will eat the food. Since in this scenario rewards consist of *empathy* = 0.7 and *needy* = 0.6, the Robot agent will not eat food.

- Consider *empathy* = 0.6 and *needy* = 0.6 Based on the reward, probability matrix, and utility, the optimal policy of the Robot agent is determined. Here, food is limited. As soon as one of the agents eats the food, the food will disappear. In this situation reward is *needy* because numerical value of *needy* is same as *empathy*. Since reward is *needy* the Robot eats food and do not help subject 1 and subject 2. In following steps, you can see that the Robot agent moves towards the food and in step 5 the Robot agent eats food. (Food is limited). In Step 6 food disappears.

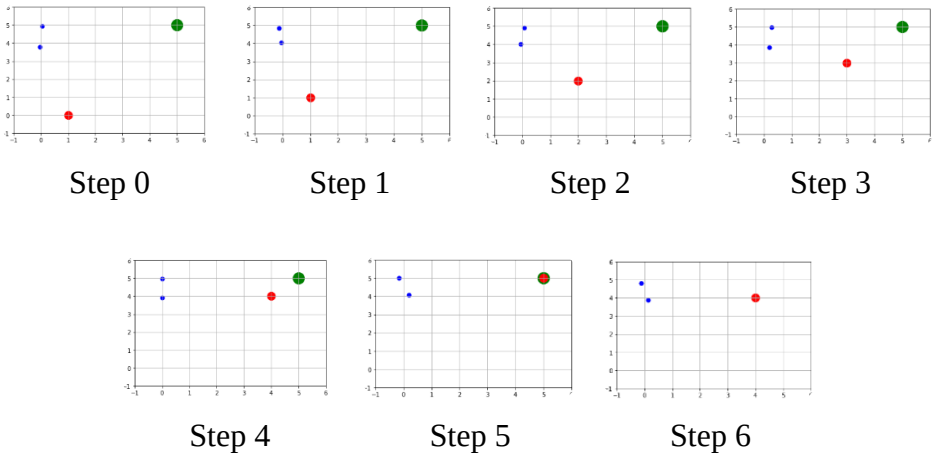

Figure 4.9 Example of CAT-MDP (Limited Food)

4.11 PROBLEM STATEMENT 2 – HAND CODED POLICY

A small grid world with one Robot cat agent and two subject 1 and subject 2 agent cats is built. The Robot agent can have varying degrees of *needy* and *empathy*. The *needy* determines how much the Robot agent wants to eat. The *empathy* determines how much the Robot agent wants the subject 1 and subject 2 to eat. Both can be positive or negative numbers. The Robot agent's policy is to be determined by Hand coded policy that uses Dijkstra's algorithm [1]. We are going to apply Hand Coded Policy to this problem and see how the Robot agent, subject 1 and subject 2 behave based on *empathy* and *needy* factor.

Hand Coded Policy for The Robot:

I implemented a hand coded policy to guide the Robot agent's behavior. I am using Dijkstra algorithm [1] to determine the optimal policy for the Robot. Based on reward (*needy* and *empathy*) we will determine optimal policy for the Robot. Depending on the value of reward (*needy* and *empathy*) we will determine if we will find shortest distance between the Robot agent and food only or if we will find shortest distance between the Robot agent and, subject 1 and subject 2 and then find shortest distance between the Robot agent and food, while subject 1 and subject 2 will follow the Robot agent. Here, depending on *needy* and *empathy* value we will call Dijkstra's algorithm with the Robot agent as source and food or subject1 and subject 2 as destination. Dijkstra's algorithm will return path from source to destination. This path is sequence of action that the Robot agent will take per time step.

Following is the pseudo code:

Algorithm 6: Hand Code Policy

```
Dijkstra algorithm returns path from source to destination node
if needy > 0 then
        if empathy < 0 or empathy =0 then
                call Dijkstra's algorithm with The Robot as source and food as destination
                feed the path i.e sequence of action that The Robot will take per time step
        else if empathy > 0 then
                call Dijkstra's algorithm with The Robot as source and subject1/subject 2 as destination
                call Dijkstra's algorithm with The Robot as source and food as destination
                feed the path i.e sequence of action that The Robot will take per time step
end if
else if needy < 0 then
        if empathy < 0 or empathy = 0 then
                The Robot stays at the same location
        else if empathy > 0 then
                call Dijkstra's algorithm with The Robot as source and subject1/subject 2 as destination
                call Dijkstra's algorithm with The Robot as source and food as destination
                feed the path i.e sequence of action that The Robot will take per time step
end if
else if needy = 0
        if empathy < 0 or empathy = 0 then
                The Robot stays at the same location
        else if empathy > 0 then
                call Dijkstra's algorithm with The Robot as source and subject1/subject 2 as destination
                call Dijkstra's algorithm with The Robot as source and food as destination
                feed the path i.e sequence of action that The Robot will take per time step
end if
```

Figure 4.10 Pseudocode for Hand-Coded Policy

4.12 EXAMPLES OF CAT- HAND CODED POLICY

- Consider *empathy* = 1 and *needy* = -1. Based on the reward and hand coded algorithm, policy of the Robot agent is determined. Since the reward is *empathy* the Robot agent decides to help subject 1 and subject 2 to eat food. In this scenario, it will help both subjects to eat food. (Food is unlimited)

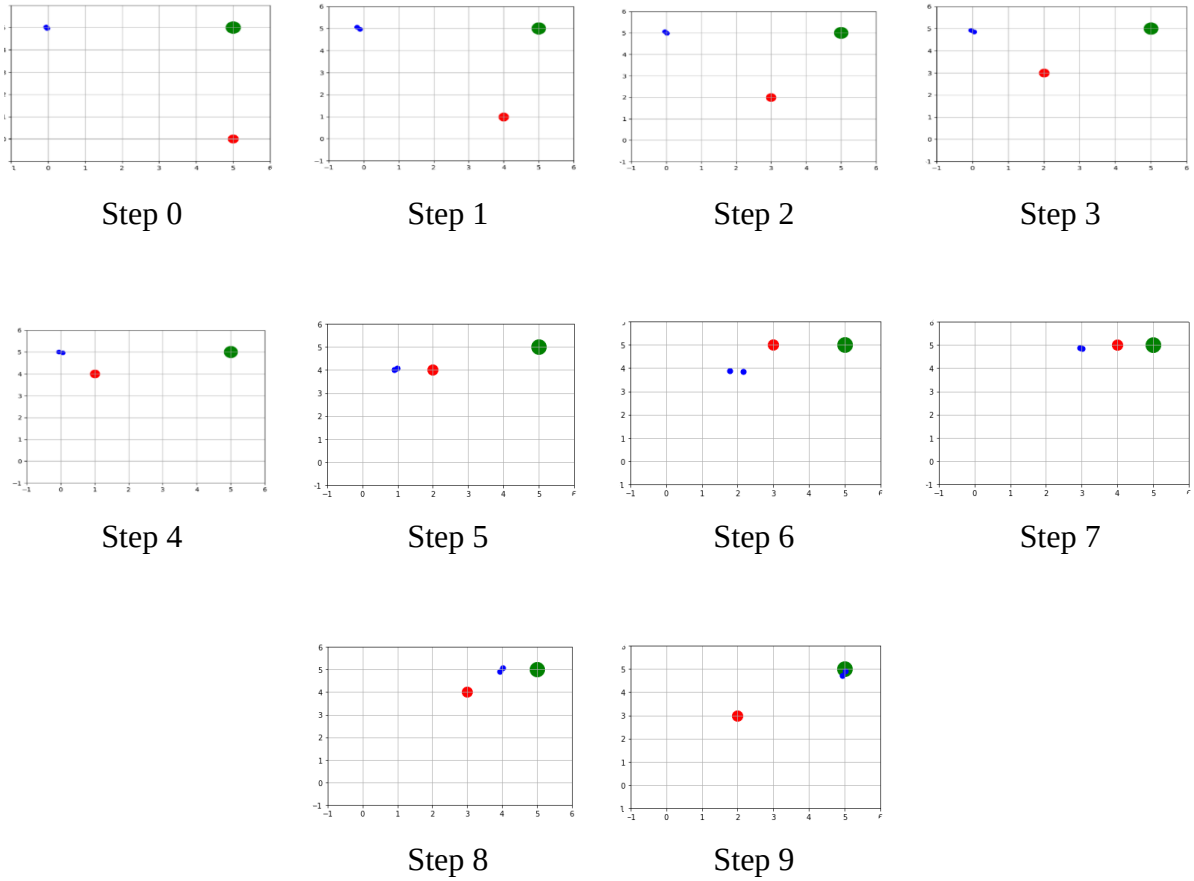

Figure 4.11 Example 1 of CAT- Hand-Coded Policy (Unlimited Food)

In Step 1,2,3,4 we can see that the Robot agent is moving towards subject 1 and subject 2 so that the Robot agent can take subject 1 and subject 2 towards food. Here, in step 5 both subject 1 and subject 2 move to the cell that the Robot agent occupied in step 4 since the

Robot agent was in subject 1 and subject 2 viewing range. In Step 6, both subject 1 and subject 2 would move to the cell that was occupied by the Robot agent in step 5. Similarly, in step 7,8 subject 1 and subject 2 would occupy the cell which was previously occupied by the Robot agent in last step. In Step 8, food is in viewing range of both subject 1 and subject 2. So, in step 9 subject 1 and subject 2 would move to the cell that has food and will eat the food. Since in this scenario rewards consist of *empathy* = 1 and *needy* = -1, the Robot agent will not eat food.

- Consider *empathy* = -1 and *needy*= 1. Based on the reward and hand coded algorithm, policy of the Robot agent is determined. Since, reward is *needy* the Robot agent eats food and do not help subject 1 and subject 2. In Following steps, you can see that the Robot agent move towards the food and in step 5 the Robot eats food. (Food is unlimited).

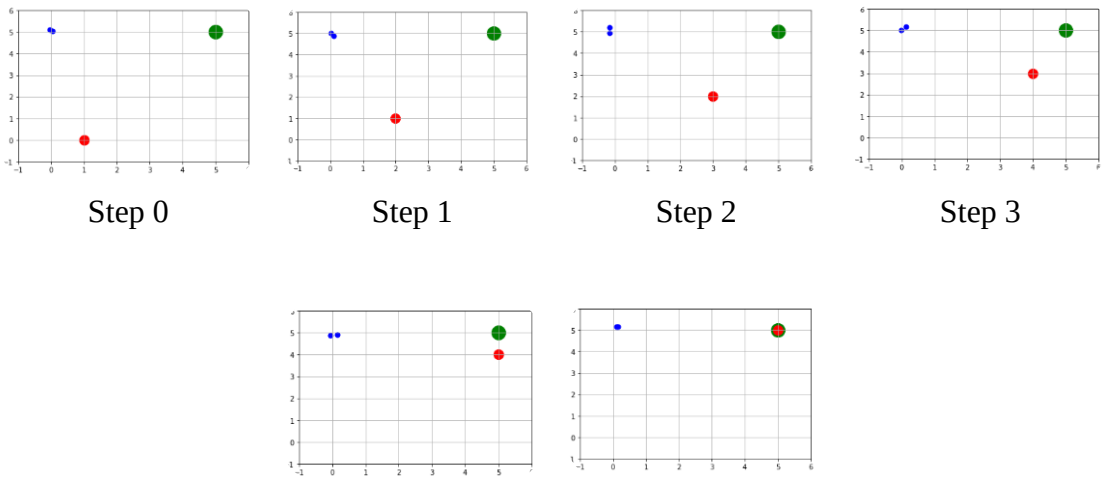

Figure 4.12 Example of CAT- Hand-Coded Policy (Unlimited Food)

Chapter 5

RESULTS

5.1 CAT- MDP (UNLIMITED FOOD)

Our problem involves a small grid world with one Robot cat and a two subject 1 and subject 2 cats. To understand impact of empathy in the behavior of 3 agents, we are varying numerical value of *empathy* and *needy* from -1 to 1 for the Robot agent. Here, the Robot agent will eat the food or help subject 1 and subject 2 to eat the food depending on numerical value of *empathy* and *needy*. Here, change is observed in the behavior of the Robot agent, subject 1 and subject 2 by measuring average distance of food from the Robot agent, subject 1 and subject 2 when numerical values of *empathy* and *needy* varies from -1 to 1 respectively.

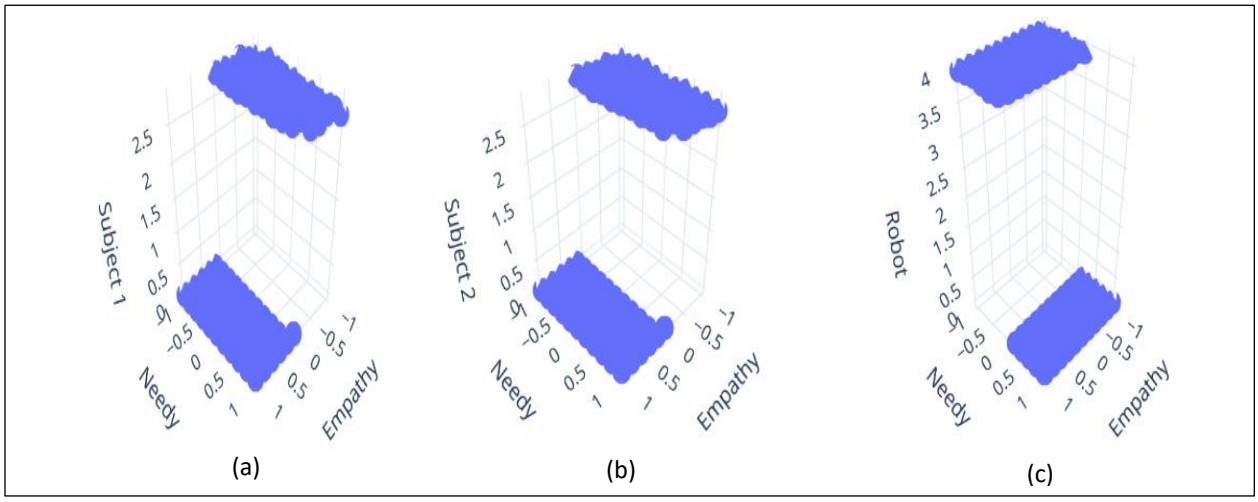

Figure 5.1 Average Distance to food vs Needy and Empathy (CAT-MDP: Unlimited Food)

The figure above shows distance of agents from food depending on *needy* and *empathy* factor. Here, numerical value of *empathy* and *needy* is varied from -1 to 1. The distance at the last time step was averaged over 30 episodes with random initial state. As you can see when *empathy* is -1

and *needy* is 1 in fig (c) average distance between the Robot cat agent and food is 0 since the Robot agent eats the food and in fig (a) and fig (b) average distance from subject 1 and subject 2 is around 3 unit since subject 1 and subject 2 doesn't eat the food since the food is out of subject 1 and subject 2's view and Robot agent avoids the subject 1 and subject 2 . Similarly, when *empathy* is 1 and *needy* is -1 in fig (c) average distance between the Robot agent and food is around 4 unit, since the Robot agent takes the subject 1 and subject 2 near the food but doesn't eat the food and in fig (a) and fig (b) average distance from subject 1 and subject 2 is around 0 unit since subject 1 and subject 2 eat the food. Here, food is unlimited.

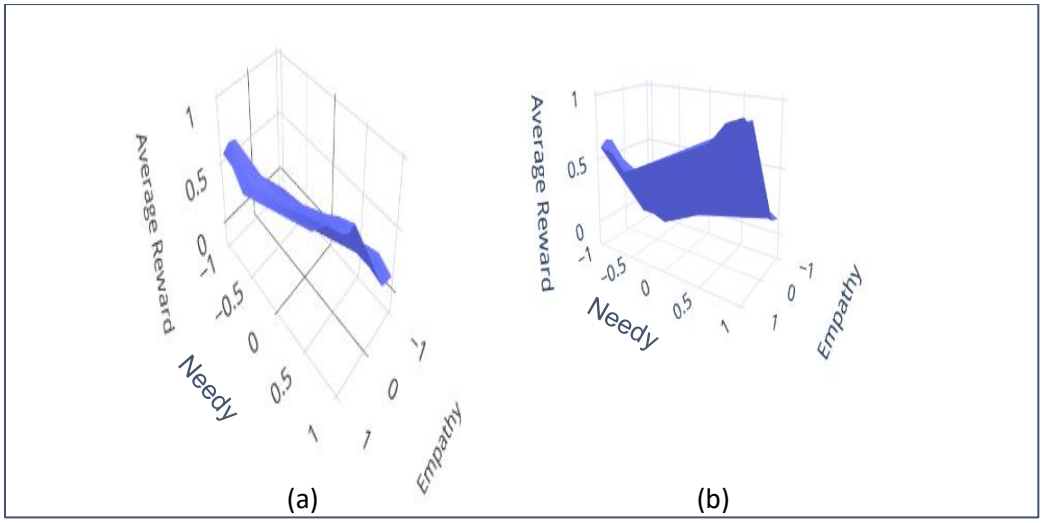

Figure 5.2 Average Reward vs Needy and Empathy (CAT-MDP: Unlimited Food)

The figure above shows Robot agent's average reward per number of steps to reach the food over 30 episodes. Here, numerical value of *empathy* and *needy* is varied from -1 to 1. As you can see when *needy* is 1 and *empathy* is 1 the average reward is maximum since the Robot agent helps both subject 1 and subject 2 to eat the food and it will also eat the food. As *empathy* decreases and *needy* increases Robot agent's average reward decreases.

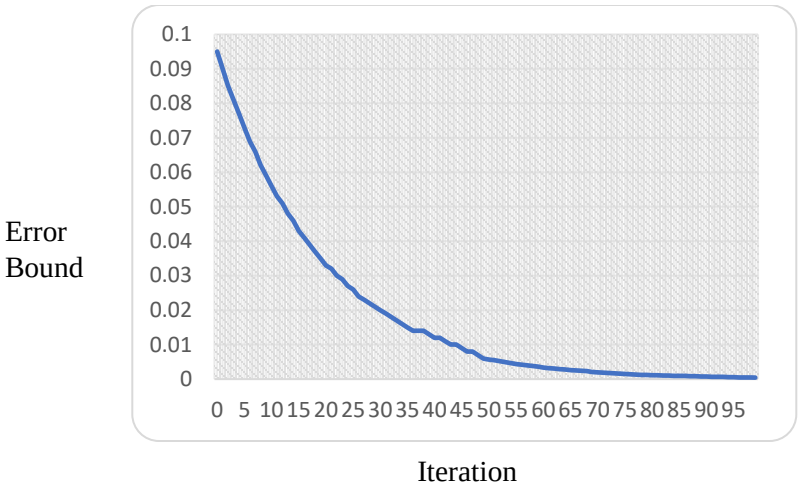

Figure 5.3 Error Bound vs Number of Iterations for Value Iteration Algorithm

The figure above shows that error reduces to 0 as the iteration increases as the agent learns the policy when using value iteration algorithim [2] [22] for CAT-MDP. Here error is difference between expected utility and new utility. Error bound is the upper bound of the difference between utilities of two consecutive iterations by $||u_{i+1}-u_i||<\epsilon\frac{1-\gamma}{\gamma}$ where γ is discount factor [19].

MDP using Policy Iteration algorithm: In this study policy iteration algorithm is implemented to get optimal policy for the Robot. The Policy iteration algorithm converges when there is no change in policy for two consecutive iterations. For our implementation policy iteration did not converge. It ran for almost 900 iterations. Different action for subsequent states between two policies of consecutive iterations keeps on fluctuating between 432,446, and 464.

5.2 CAT- MDP (LIMITED FOOD)

Our problem involves a small grid world with one Robot cat and two subject 1 and subject 2 cats. To understand impact of empathy in the behavior of 3 agents, numerical value of *empathy* and *needy* is varied from -1 to 1 for the Robot agent. The food disappears when any one of agent eats the food. Here, the Robot agent will eat the food or help either subject 1 or subject 2 to eat the food depending on numerical value of *empathy* and *needy*. Here change is observed in the behavior of the Robot agent, subject 1 and subject 2 by measuring average distance of food from the Robot agent, subject 1 and subject 2 when numerical values of *empathy* and *needy* varies from -1 to 1 respectively.

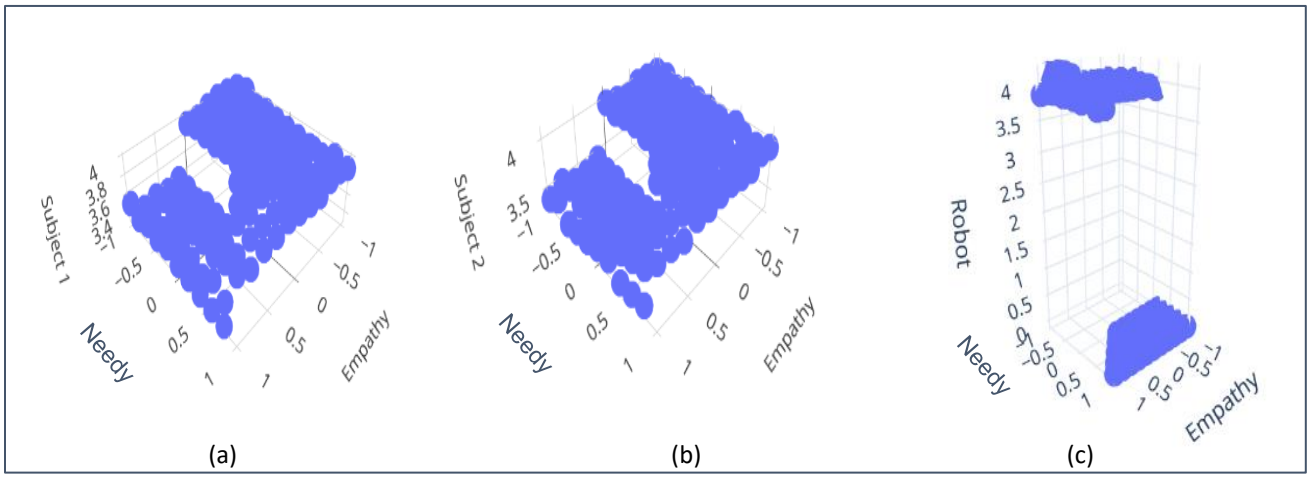

Figure 5.4 Average Distance to food vs Needy and Empathy (CAT-MDP: Limited Food)

The figure above shows distance of agents from food depending on *empathy* and *needy* factor. Here, numerical value of *empathy* and *needy* is varied from -1 to 1. The distance at the last time step was averaged over 30 episodes with random initial state. For this experiment food is limited. Only one agent from the Robot agent, subject 1 and subject 2 can eat food. As soon as one agent eats food, the food will disappear. As you can see when *empathy* is -1 and *needy* is 1 in fig (c) average distance between the Robot agent and food is 0 since the Robot agent eats the food and

in fig (a) and fig (b) average distance from subject 1 and subject 2 is around 3.5 unit since subject 1 and subject 2 doesn't eat the food since the food is out of subject 1 and subject 2's view and the Robot agent avoids the subject 1 and subject 2. Similarly, when *empathy* is 1 and *needy* is -1 in fig (c) average distance between the Robot agent and food is 4 unit since the Robot agent help either subject 1 or subject 2 to eat the food depending on number of time step it will take to help that subject to reach near the food and in fig (a) and fig (b) average distance to food from subject 1 and subject 2 is around 3.5 unit since either subject 1 or subject 2 eat the food. If numerical value of *needy* is greater than numerical value of *empathy* by 0.1 or more, then the Robot agent will eat the food. If numerical value of *empathy* is greater than numerical value of *needy* by 0.1 or more, then either subject 1 or subject 2 will eat the food.

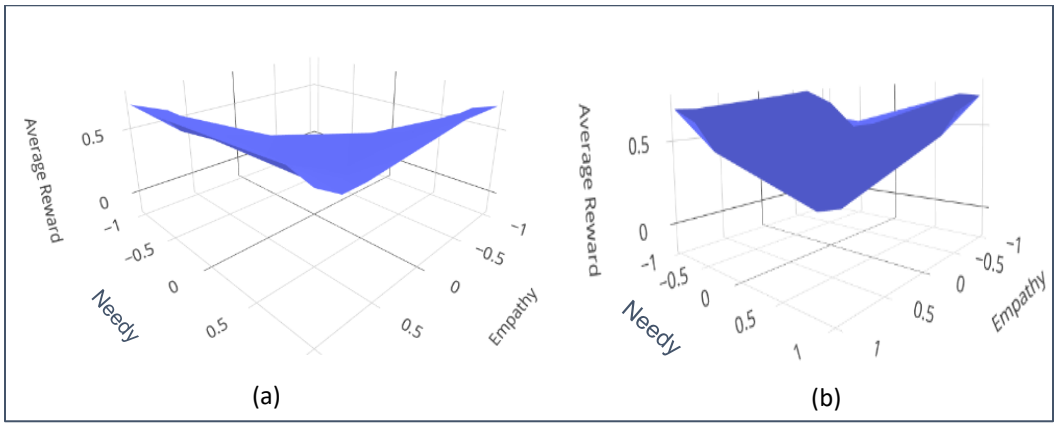

Figure 5.5 Average Reward vs Needy and Empathy (CAT-MDP: Limited Food)

The figure above shows Robot agent's average reward per number of steps to reach the food over 30 episodes. Here, numerical value of *empathy* and *needy* is varied from -1 to 1. For this experiment food is limited. Only one agent from the Robot agent, subject 1 and subject 2 can eat the food. As soon as one agent eats food, the food will disappear. Here, since food disappears as soon as any one of 3 agent eats food, the Robot agent's reward is either numerical value of *needy* or *empathy*. As, you can see in above fig when either *needy* is 1 or *empathy* is 1 or both *empathy*

and *needy* is 1 the average reward is maximum since the Robot agent gets positive reward for eating food or helping either subject 1 or subject 2 to eat food respectively.

5.3 CAT-HAND CODED POLICY (UNLIMITED FOOD)

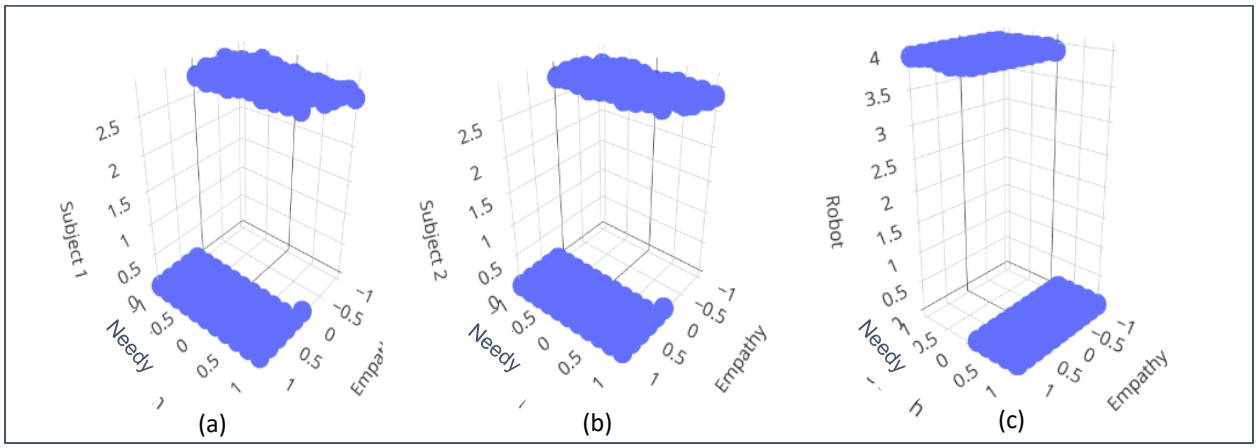

Figure 5.6 Average Distance to food vs Needy and Empathy (Unlimited Food)

The figure above shows average distance of agents from food depending on *needy* and *empathy* factor. Here, food is unlimited. Here, numerical value of *empathy* and *needy* is varied from -1 to 1. The distance at the last time step was averaged over 30 episodes with random initial state. As you can see when *empathy* is -1 and *needy* is 1 in fig (c) average distance between the Robot agent and food is 0 since the Robot agent eats the food and in fig (a) and fig (b) average distance from subject 1 and subject 2 is around 3 unit since subject 1 and subject 2 doesn't eat the food since the food is out of subject 1 and subject 2's view and Robot agent avoids the subject 1 and subject 2. Similarly, when *empathy* is 1 and *needy* is -1 in fig (c) average distance between the Robot agent and food is around 4 unit, since the Robot agent takes the subject 1 and subject 2 near the food but doesn't eat the food and in fig (a) and fig (b) average distance to food from subject 1 and subject 2 is around 0 unit since subject 1 and subject 2 will eat the food.

Chapter 6

CONCLUSION & FUTURE WORK

After doing multiple simulations for both CAT-MDP and Hand Coded Policy, in this study it is found that CAT- MDP works like Hand Coded-Policy for small world environment when food is unlimited. But in MDP agents learn whereas Hand Coded Policy is hard-coded. When food is unlimited behavior of the Robot agent changes based on positive, negative or 0 value of *empathy* and *needy*. When food is limited, we can see that behavior of the Robot agent changes for small change in numerical value of *empathy* and *needy*. If positive numerical value of *empathy* is greater than positive numerical value of *needy* than behavior of the Robot agent is empathetic and when positive numerical value of *empathy* is less than positive numerical value of *needy* than behavior of the Robot agent is needy.

My next step would be to test both the methods on large environments and see if there is any change in result. After that I am planning to implement algorithms for Markov Game [12] and then compare the result with the CAT-MDP results. Markov game includes the framework that have multiple adaptive agents dependent on the actions chosen jointly by the set of interacting or competing goals [12]. Challenges in implementing Markov game is to figure out reward function for both the agents if they work together for end goal. One of the issues is the zero sum issue where two agent who work together for end goal with total sum of loss and gain between two interacting agents is zero. It is difficult for our model to implement with Markov zero sum game.

Other Markov game theory concept which I like to understand and see if it can implement for our problem is Nash Equilibrium [23].

I would also like to implement Q-learning [24] instead of value iteration and then compare the result with the CAT-MDP results. In Q-learning, agent learns from action value function that for given action taken in a given state predicts expected utility. Q-learning does not require model of the environment. In Q-learning policy is not needed, it will determine its optimal policy by taking random action. Q-learning will take large of num of iteration to converge. I am planning to observe behavior of agent when we implement Q learning. Challenge while working with Q-learning is to converge with a smaller number of iterations.

Next step would be to try POMDP (Partially Observable Markov Decision Process, compare it with CAT-MDP and Hand Coded Policy. A POMDP is basically Markov Decision Process with partial observability [25]. In a POMDP we limit how far an agent can observe the environment from given state. Since agent cannot see all state of environment, we need to remember history of agent's decision process which makes it more complex than MDP. Next step would be to understand Inverse Reinforcement Learning [16] and Plan Recognition [21]and implement empathy in our environment.

APPENDIX

- https://github.com/Dhwani4/Master-Thesis

REFERENCES

[1] Dijkstra, Edsger W. "A note on two problems in connexion with graphs." *Numerische mathematik* 1.1 (1959): 269-271.

[2] Bellman, Richard. "A Markovian decision process." *Journal of mathematics and mechanics* (1957): 679-684.

[3] Leite, Iolanda, et al. "The influence of empathy in human–The Robot relations." *International journal of human-computer studies* 71.3 (2013): 250-260.

[4] Castellano, Ginevra, et al. "Towards empathic virtual and The Robotic tutors." *International Conference on Artificial Intelligence in Education*. Springer, Berlin, Heidelberg, 2013.

[5] Zhou, Li, et al. "The design and implementation of XiaoIce, an empathetic social chatbot." *arXiv preprint arXiv:1812.08989*(2018).

[6] Al-Shihabi, Talal, and Ronald R. Mourant. "A framework for modeling human-like driving behaviors for autonomous vehicles in driving simulators." *Proceedings of the fifth international conference on Autonomous agents*. ACM, 2001.

[7] De Carolis, Berardina Nadja, et al. "Towards an empathic social The Robot for ambient assisted living." *ESSEM@ AAMAS*. 2015.

[8] Šabanović, Selma, et al. "PARO Robot affects diverse interaction modalities in group sensory therapy for older adults with dementia." *2013 IEEE 13th International Conference on Rehabilitation The Robotics (ICORR)*. IEEE, 2013.

[9] Shibata, Takanori. "Therapeutic seal The Robot as biofeedback medical device: Qualitative and quantitative evaluations of The Robot therapy in dementia care." *Proceedings of the IEEE*100.8 (2012): 2527-2538.

[10]Oh, Jae C. "Emergence of kin habitats and cooperation in multi-agent environment." *Proceedings of the 2nd international workshop on frontiers in evolutionary algorithms (FEA'98), Research Triangle Park, NC*. 1998.

[11]Dorigo, Marco, and Mauro Birattari. *Ant colony optimization*. Springer US, 2010.

[12]Littman, Michael L. "Markov games as a framework for multi-agent reinforcement learning." *Machine learning proceedings 1994*. Morgan Kaufmann, 1994. 157-163.

[13]Lanctot, Marc, et al. "A unified game-theoretic approach to multiagent reinforcement learning." *Advances in Neural Information Processing Systems*. 2017.

[14]Armstrong, Karen. *A history of God: The 4,000-year quest of Judaism, Christianity and Islam*. Ballantine Books, 2011.

[15]Davis, Mark H. Empathy: A social psychological approach. Routledge, 2018.

[16]Natarajan, Sriraam, et al. "Multi-agent inverse reinforcement learning." *2010 Ninth International Conference on Machine Learning and Applications*. IEEE, 2010.

[17]https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html#scipy.sparse.csr_matrix

[18]https://www.merriam-webster.com/dictionary/empathy

[19]Dr. Katz's course material and slides

[20]https://www.inc.com/justin-bariso/there-are-actually-3-types-of-empathy-heres-how-they-differ-and-how-you-can-develop-them-all.html

[21]Litman, Diane J., and James F. Allen. "A plan recognition model for subdialogues in conversations." *Cognitive science*11.2 (1987): 163-200.

[22]Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[23]Myerson, Roger B. "Refinements of the Nash equilibrium concept." *International journal of game theory* 7.2 (1978): 73-80.

[24]Watkins, Christopher JCH, and Peter Dayan. "Q-learning." *Machine learning* 8.3-4 (1992): 279-292.

[25]Paquet, Sébastien, Ludovic Tobin, and Brahim Chaib-Draa. "An online POMDP algorithm for complex multiagent environments." *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. ACM, 2005.

Dhwani Patel

VITA

Skills

HW Description Languages: Verilog, VHDL, System Verilog

Programming Languages: C/C++, Core Java, Embedded C

OS Internals: Linux, Windows

Scripting: Perl, Python

EDA Tools: Cadence, MATLAB, Xilinx, LabVIEW, Multisim, OrCAD, ICAP4, Logic Work 4.0, PSpice

Education Background

MS in Computer Science, Syracuse University **(Aug 17-Present)** GPA:3.6

MS in Computer Engineering, University of Massachusetts, Lowell GPA:3.6

(Date of Completion: May 9th 2015)

(Date Degree was received: May 16th 2015)

BE in Electronics and Communication, Gujarat Technological University, India GPA:8.2

(Date of Completion: May 2013)

(Date Degree was received: 16th Jan 2014)

Work Experience/Internship

Analog Devices Inc. (May'14-Aug'14):

As a graduate intern, Evaluated and validated ClioSoft tool for various IP reuse Methodology. Improved efficiency by 30% in IP search and consolidation.

Volansys Technologies. (May'12- May'13):

As an undergrad intern, Used DM368 video processor based on ARM9 architecture to facilitate secure interface between 'Outdoor Media Player' and advertisement billboard.

Academic Experience (Syracuse University)

Teaching Assistant (Aug'18-Dec'18):

Assisted undergraduate students with programming in python. Working as teaching assistant for Introduction to Computing course.

Master's Thesis (Jan 19-Aug 19):

Empathy Based Reinforcement Learning.

Academic Experience (University of Massachusetts, Lowell)

Research Assistant (Feb'14- May'14):

Analyzed the causes of increase in the leakage in Low power VLSI Circuits. Proposed mitigation techniques minimizing the leakage to the maximum threshold level.

Teaching Assistant (Jan'15 – May'15):

Assisted graduate students in IC fabrication labs. Worked as a graduate teaching assistant for Logic Design course.